MongoDB – Modeling and querying a Document store

We will be using MongoDB as our Document store.
On the mongodb.com Web site, download the community server version (version greater than 4.4). Be careful to take a version that corresponds to your distribution (you may have bad surprises with the first choice on the list: Amazon Linux). Continue with the installation by unzipping the tgz file. You must have a directory of the following type (for version 4.4.1):
`mongodb-linux-x86_64-ubuntu2004-4.4.1`

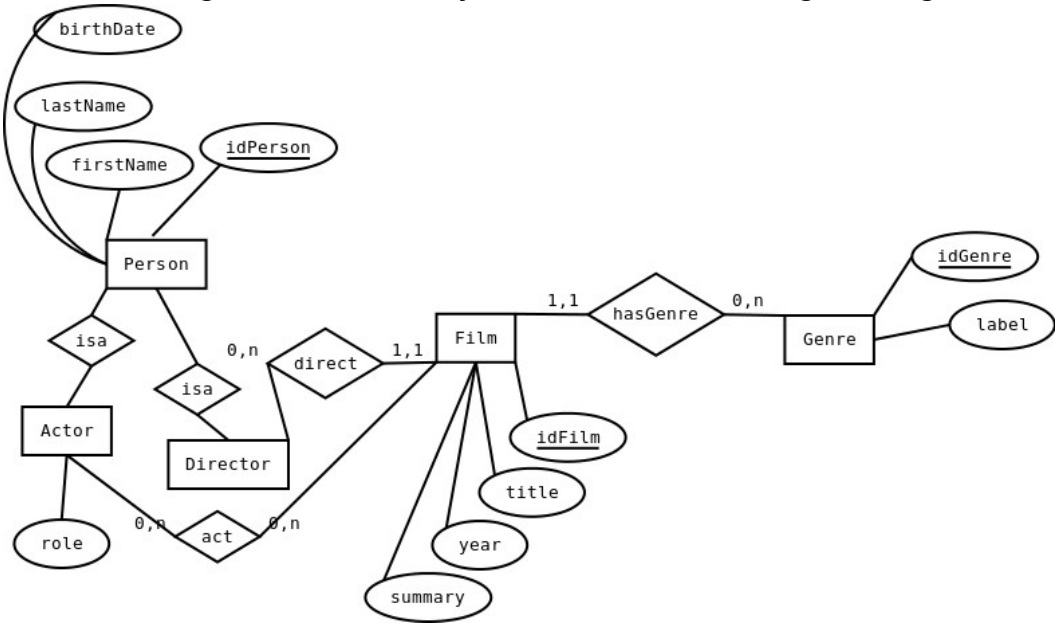
In the bin directory of your mongodb installation, there are several executable files that we will use during this session. MongoDB's execution mode follows aclient / server type. By default, the server listens on port 27017 (and on 28017 for administrative information).
It is first necessary to create a directory (e.g. data in your current directory or in the / tmp directory on the university machines). We start the server with the executable (mongo daemon - `mongod`) indicating the directory where the data will be located: `mongod --dbpath` *`repCourant/data`*

From another console, the MongoDB shell is launched with the `mongo` executable. It is a javascript shell (therefore possible interpretation of the code in this programming language) which allows the database to be administered and queries to be launched.

Shell
The `help` command is used to obtain a list of available commands: `show dbs`, `show collections`, `use <dbName>` (to indicate the database with which you want to work on).

JSON
The MongoDB NoSQL database management system manages JavaScript Object Notation (JSON) documents. It is a text data format derived from the object notation of the JavaScript language consisting of key/value pairs and arrays.

Modeling
As part of the modeling of a video library, we have the following ER diagram.



We want to implement a similar database on MongoDB. You must take into account that a document store is generally schemaless and that its query language (if there is one) does not allow to represent joins. We must therefore think in terms of **denormalization** as soon as the modeling stage.

1. In this question, we consider 2 different use cases of the document-oriented database translating the above entity association diagram. For each model, set up the collection with the following film (document):
The movie is "A History of Violence", year "2005", its summary "Tom Stall, a humble family man and owner of a popular neighborhood restaurant, lives a quiet but fulfilling existence in the Midwest. One night Tom foils a crime at his place of business and, to his chagrin, is plastered all over the news for his heroics. Following this, mysterious people follow the Stalls' every move, concerning Tom more than anyone else. As this situation is confronted, more lurks out over where all these occurrences have stemmed from compromising his marriage, family relationship and the main characters' former relations in the process.". It is an american movie (country USA) and its genre is ''Crime''.
The director is David Cronenberg who was born in 1943. The main actors of the movie are Ed Harris (born in 1950) plays the role of "Carl Fogarty", Vigo Mortensen (born in 1958) plays the role of "Tom Stall", Maria Bello (born in 1967) plays the role of "Eddie Stall " and William Hurt (born in 1950) as "Richie Cusack".

a. Considering that the application exploiting the database requests essentially on films and not on actors and directors, propose a representation using the JSON format for a document containing the History of Violence information and implement it on a new database (dbma) .
b. In another use case, the users of the application search both by movie and by actors. In the movie-driven query, the users expect all the movie information (including actors, directors, etc.). In the actor-driven query, the users expect the name of movies they are playing in. You also need to support a solution to retrieve full movie information from an actor-driven query (that would be implemented in an application). Propose a representation in JSON format for that database based on the «A history of violence» movie data.

2. Based on the database modeling of 1a. Store the following movie
"The Social network", 2010, Drama, "On a fall night in 2003, Harvard undergrad and computer programming genius Mark Zuckerberg sits down at his computer and heatedly begins working on a new idea. In a fury of blogging and programming, what begins in his dorm room soon becomes a global social network and a revolution in communication. A mere six years and 500 million friends later, Mark Zuckerberg is the youngest billionaire in history... but for this entrepreneur, success leads to both personal and legal complications.", USA. Actors are: Jesse Eisenberg (born in 1983) plays the role of Mark Zuckerberg", Rooney Mara (1985) plays the role of Erica Albrig. The director is David Fincher (1962)

3. Given this database:
a. Display the 2005 movies. Between the parentheses of `find()`, we place a filter in JSON format: `{year: '' 2005 ''}`. Make it more readable with `pretty()`.
b. Provide all the titles of the 2005 films. To select only a few output attributes, we add a filter in the second position of `find()`. This filter includes a key / value pair of the attribute type: {0,1} with 1 if you want the display, 0 otherwise.
c. We still get _id. How to remove it with what we have just seen.
d. Display the summary of «The Social network».
e. Retrieve the director of «A History of Violence».
f. retrieve the actors of «The Social network»
g. Display the list of movies where Cronenberg is the director's name. So this

amounts to filtering in a sub document (director). For this, we use the object form that we use in javascript: keyDocument.attribut: director.last_name: '' Cronenberg ''. Display the names (last name) of directors whose first name is David.

i. Display the list of films where "Vigo" is the first name of an actor. More difficult because we filter a sub-document, element of a table. We use for that `$elemMatch` in the form: `{arrayName: {$ elemMatch: {key: value, ..}}}`

j. Operators for comparisons: `$gt, $gte, $lt, $lte` with syntax of the form: `{key: {$ gte: value}}`. Display movies whose year is> = to 2005.

k. Retrieve the names of the directors that were born after 1960.

l. With the `$exists` operator we can know if a key is in a document. This is convenient since MongoDB is a schemaless store. For example to know if there is a '' country '' key in a document: `{country: {$ exists: 1}}` (or true instead of 1). Test for the absence, we replace 0 (false) by 1. Display the names of the films for which there is a director.

m. There are other operators with '$', for example $ in, `$nin` (know if a value is in an array), `$all` (all values are in an array), `$type` (test on the type of a value), `$not, $where, $or` and `$and`. For example, American films or like '' Drama '': `db.movies.find ({$ or: [{country: "USA"}, {"genre": "Drama"}]}, {title: 1})`
Display films of the genre Drama or those with a summary.

n. You insert a new 2007 movie called “Into the Wild”, Drama directed by “Sean Penn”. Check how many doc there is in the database.

o. You need to add data on Into The wild: the summary: "After graduating from Emory University, top student and athlete Christopher McCandless abandons his possessions, gives his entire $ 24,000 savings account to charity and hitchhikes to Alaska to live in the wilderness." Along the way, Christopher encounters a series of characters that shape his life. ". On `db.movies.update ({id du film}, {summary: '' .. ''})`. Display the whole movie information
A JavaScript-style solution: `var into = db.movies.findOne ({title: '' Into the Wild ''})` and now we add the attributes we want: into.summary = '' After .. '' . We type *into* to verify. Add an empty array for actors. We validate by doing `db.movies.update ({idFilm}, into)`

p. Remove the actor key in that movie: `db.movies.update({..},{$unset : {actors:1}})`

q. We now add actors to the Into the Wild movie: `db.movies.update({"_id" : ObjectId(`*yourLocalId*`)},{$addToSet:{actors: {"last_name":"Hirch","first_name":"Emile"}}})`. Also add William Hurt in that movie.

r. To remove a whole document : `db.movies.remove({id})`. Remove the Into the Wild movie